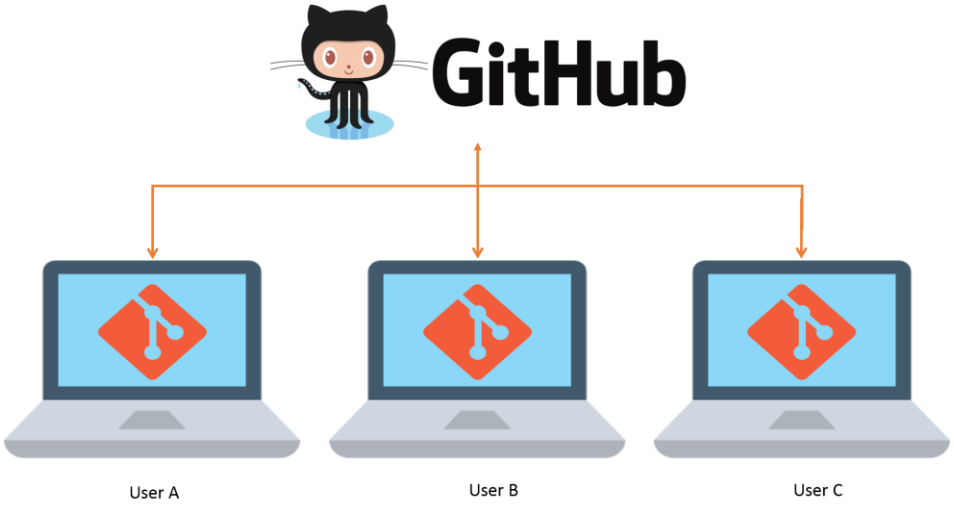
Angel Yeremi Ledesma Ayala

Git y GitHub

Introducción a Git

Git es un sistema de control de versiones distribuido ampliamente utilizado que facilita el seguimiento de cambios en proyectos de software y la colaboración entre desarrolladores. Fue creado por Linus Torvalds en 2005 y se ha convertido en una herramienta fundamental en el desarrollo de software.



- **¿Qué es un Sistema de Control de Versiones (SCV)?**

Un Sistema de Control de Versiones (SCV) es una herramienta que permite gestionar cambios en el código fuente y otros archivos. Su función principal es llevar un registro de las modificaciones realizadas en un proyecto a lo largo del tiempo. Esto es esencial en el desarrollo de software para:

1. **Seguimiento de Cambios:** Un SCV registra quién hizo qué cambio, cuándo y por qué. Esto facilita la resolución de problemas y la colaboración en proyectos de desarrollo.
2. **Restauración de Versiones Anteriores:** Puedes regresar a versiones anteriores del proyecto en caso de que ocurra un error o necesites una versión anterior funcional.
3. **Colaboración:** Permite a múltiples desarrolladores trabajar en un proyecto de manera simultánea, fusionando sus cambios de manera controlada.
4. **Respaldo y Seguridad:** Los SCV almacenan copias de seguridad del proyecto en servidores o repositorios, lo que garantiza la seguridad de los datos.

- **¿Por qué Git?**

Git se destaca como uno de los sistemas de control de versiones más populares y versátiles por varias razones:

1. **Distribución:** Git es un SCV distribuido, lo que significa que cada desarrollador tiene una copia completa del historial del proyecto en su máquina local. Esto permite

trabajar sin conexión y facilita la colaboración sin depender de un único repositorio central.

2. **Velocidad:** Git es increíblemente rápido, lo que lo hace eficiente para gestionar proyectos de cualquier tamaño.
3. **Ramas (Branches):** Git permite trabajar en ramas separadas, lo que facilita el desarrollo de nuevas características sin afectar la rama principal del proyecto.
4. **GitHub:** GitHub es una plataforma web que se integra perfectamente con Git y permite alojar repositorios de Git en línea. Es ampliamente utilizado para la colaboración en proyectos de código abierto y privados.
5. **Amplia Comunidad:** Git cuenta con una comunidad activa de desarrolladores y una abundancia de recursos en línea, lo que facilita su aprendizaje y resolución de problemas.

- **Conceptos Clave de Git**

Para comprender Git, es importante familiarizarse con algunos conceptos clave:

1. **Repositorio (Repository):** Un repositorio es un almacén de archivos y carpetas que están bajo el control de Git. Puede ser local o remoto.
2. **Commit:** Un commit es una instantánea de los cambios en el proyecto en un momento dado. Cada commit tiene un mensaje descriptivo.
3. **Rama (Branch):** Una rama es una línea de desarrollo independiente en un proyecto. Puedes trabajar en múltiples ramas para desarrollar nuevas características o solucionar problemas sin afectar la rama principal.
4. **Pull Request (PR):** En GitHub, un Pull Request es una solicitud para fusionar cambios desde una rama secundaria en la rama principal del proyecto. Es una característica esencial para la colaboración.
5. **Conflictos (Conflicts):** Los conflictos ocurren cuando dos o más desarrolladores modifican el mismo archivo en diferentes ramas y Git no puede determinar automáticamente cómo fusionar esos cambios.
6. **Merge:** Fusionar es el proceso de combinar los cambios de una rama en otra. Esto se hace generalmente a través de un Pull Request.

Comandos Básicos de Git

- **git init**: Este comando se utiliza para crear un nuevo repositorio Git en un directorio. Inicializa un repositorio vacío y comienza a rastrear los archivos en ese directorio.

- **git clone**: Si deseas obtener una copia de un repositorio Git existente, puedes usar el comando "git clone". Proporciona una URL del repositorio y descargará una copia local completa, incluyendo todos los archivos y el historial de cambios.

- **git add**: Antes de confirmar cambios en Git, debes agregar los archivos modificados al área de preparación (staging area) con el comando "git add". Esto prepara los archivos para su inclusión en el próximo commit.

- **git commit**: El comando "git commit" toma los archivos en el área de preparación y crea una instantánea del estado actual de los archivos. Debes proporcionar un mensaje descriptivo que explique los cambios realizados en este commit.

- **git status**: Para obtener información sobre el estado actual de tu repositorio, puedes usar "git status". Este comando muestra los archivos modificados, los archivos en el área de preparación y otros detalles importantes.

- **git log**: El comando "git log" muestra el historial de commits en tu repositorio. Proporciona información sobre quién realizó los cambios, cuándo se realizaron y los mensajes de commit asociados.
- **git pull**: Cuando trabajas en un proyecto colaborativo y deseas obtener las últimas actualizaciones del repositorio remoto, puedes utilizar "git pull". Esto descargará los cambios más recientes y fusionará automáticamente tu trabajo local con los cambios remotos.

- **git push**: Para enviar tus cambios locales a un repositorio remoto, utiliza el comando "git push". Esto sincronizará tu trabajo local con el repositorio compartido para que otros puedan acceder a tus cambios.

- **git branch**: Git permite trabajar en múltiples ramas (branches) para desarrollar nuevas características o solucionar problemas sin afectar la rama principal. Puedes utilizar "git branch" para listar, crear y cambiar entre ramas.

- **git merge**: Cuando deseas combinar el trabajo de una rama con otra (por ejemplo, fusionar una rama de características en la rama principal), puedes usar "git merge". Este comando integra los cambios de una rama en otra.

- **git reset**: Si necesitas deshacer cambios o retroceder a un estado anterior, puedes utilizar "git reset". Hay varias opciones disponibles, como --soft, --mixed y --hard, que determinan la forma en que se deshacen los cambios.

- **git stash**: A veces, es útil guardar temporalmente cambios en progreso sin realizar un commit. Puedes utilizar "git stash" para ocultar tus cambios y recuperarlos más tarde.

Trabajando en GitHub con Repositorios (Branches, Merge, Conflicts)

GitHub es una plataforma web ampliamente utilizada para alojar repositorios de Git y facilitar la colaboración en proyectos de desarrollo de software. En esta sección, exploraremos

cómo trabajar en GitHub con repositorios, hacer uso de ramas (branches), resolver conflictos y fusionar cambios de manera efectiva.

- **Creación de un Repositorio en GitHub**

Para comenzar a trabajar en GitHub, los usuarios deben crear un repositorio en la plataforma. Los pasos incluyen:

1. Iniciar sesión en una cuenta de GitHub existente o crear una nueva cuenta si aún no se dispone de una.
2. Hacer clic en el botón "New" o "Nuevo" en la página principal de GitHub.
3. Completar los detalles del repositorio, incluyendo el nombre, descripción y opciones de visibilidad.
4. Opcionalmente, se puede agregar un archivo README, un archivo .gitignore y una licencia al repositorio.
5. Finalmente, hacer clic en el botón "Create repository" o "Crear repositorio" para crear el repositorio en GitHub.

- **Clonar un Repositorio de GitHub**

Una vez que se ha creado un repositorio en GitHub, los usuarios pueden clonarlo en su máquina local para comenzar a trabajar en él. El proceso implica el siguiente comando:

```
git clone URL_del_repositorio
```

Reemplazar "URL_del_repositorio" con la URL del repositorio en GitHub. Esto creará una copia local del repositorio en la máquina del usuario.

- **Trabajar con Ramas (Branches)**

Las ramas (branches) son esenciales en el trabajo colaborativo y en el desarrollo de nuevas características. Los usuarios pueden crear y gestionar ramas en GitHub de la siguiente manera:

- Crear una nueva rama:

```
git branch nombre_de_la_rama
```

- Cambiar a una rama existente:

```
git checkout nombre_de_la_rama
```

- Crear una nueva rama y cambiar a ella:

```
git checkout -b nombre_de_la_rama
```

- **Realizar Cambios y Commits en GitHub**

Después de clonar un repositorio desde GitHub, los usuarios pueden realizar cambios en sus archivos locales y registrar esos cambios con commits. Los pasos son:

1. Realizar cambios en los archivos locales.

2. Añadir los archivos modificados al área de preparación (staging area) con:

```
git add nombre_del_archivo
```

o

```
git add .
```

para añadir todos los archivos modificados.

3. Realizar un commit con un mensaje descriptivo:

```
git commit -m "Mensaje descriptivo de los cambios"
```

4. Subir los cambios al repositorio de GitHub:

```
git push nombre_remoto nombre_rama
```

- **Resolución de Conflictos**

Los conflictos surgen cuando dos o más desarrolladores modifican la misma parte de un archivo en diferentes ramas, y Git no puede fusionar automáticamente los cambios. Para resolver conflictos:

1. Actualizar la rama local con los cambios más recientes del repositorio remoto:

```
git pull nombre_remoto nombre_rama
```

2. Git mostrará los archivos con conflictos. Se deben editar manualmente estos archivos para resolver los conflictos.
3. Después de resolver los conflictos, añadir los archivos modificados al área de preparación:

```
git add nombre_del_archivo
```

4. Realizar un nuevo commit para finalizar la resolución de conflictos.
5. Subir los cambios resueltos al repositorio remoto:

```
git push nombre_remoto nombre_rama
```

- **Fusionar Cambios (Merge) en GitHub**

Una vez que se han completado los cambios en una rama y se desean integrar en la rama principal o en otra rama, se puede utilizar la función de fusión (merge) en GitHub:

1. Crear un Pull Request (PR) desde la rama con los cambios hacia la rama de destino.
2. Los revisores pueden revisar los cambios y, si todo está correcto, aprobar el PR.
3. Luego, se pueden fusionar los cambios en GitHub.

- **Colaboración en Proyectos de Código Abierto**

GitHub es ampliamente utilizado para proyectos de código abierto. Los colaboradores pueden seguir estos pasos:

1. Encontrar un proyecto de código abierto de interés en GitHub.
2. Realizar un "fork" del proyecto para crear una copia propia.
3. Clonar el repositorio "forked" en la máquina local.
4. Realizar cambios y crear PRs para el proyecto original.